A NEW LIBRARY TO IMPROVE TOUGH PARALLEL DEVELOPMENT

Noel Keen, George Pau, Jeff Johnson, Eric Sonnenthal, Stefan Finsterle

Lawrence Berkeley National Laboratory
Earth Sciences Division, MS 74-0120
Berkeley, CA 94720
e-mail: NDKeen@lbl.gov

ABSTRACT

Many users and developers have contributed to the success of the TOUGH suite of codes. However, the wide range of problems the codes can simulate has led to significant modification and fragmentation of the codes, exemplified by the multiple flavors of TOUGH. Maintenance of the codes becomes increasingly difficult, and improvement in one branch of the codes cannot be easily propagated to other branches. In addition, parallel computing resources cannot be uniformly utilized by all the codes, due to the lack of a parallel programming framework that simplifies this task.

We are in the process of re-engineering the TOUGH codes, specifically by developing a library (named *toughlib*), which includes functions that are commonly used by the TOUGH codes, such as the linear solvers. For example, adding a new parallel linear solver package (such as PETSc) to *toughlib* would enable any TOUGH code to use it. We are also writing functions that hide more complex constructs such as parallelism behind an abstraction layer, which will make it easier to combine serial and parallel versions of TOUGH codes. When complete, the *toughlib* library will serve as a working Application Programming Interface (API), so that all TOUGH codes using it will have similar structure. We will demonstrate how these new concepts are used in a combined version of TOUGH2 and TOUGH2-MP.

In addition, we are improving the software engineering processes that are used to develop the TOUGH codes. In particular, we have improved the build process using CMake, which simplifies the installation of external packages on multiple platforms. We also utilize the Bitbucket service, which provides maintenance-free access to the Mercurial version control system, to manage the development of this new library.

INTRODUCTION

The TOUGH suite of simulators (http://esd.lbl.gov/TOUGH) is one of the most complete software packages in the market for numerical modeling of nonisothermal multiphase flow and reactive transport in porous media. Over the past 30 years, the suite has been widely adopted by universities, government organizations, and private industry for applications to nuclear waste disposal, environmental remediation problems, energy production from geothermal, oil, and gas reservoirs as well as gas hydrate deposits, geological carbon sequestration, vadose zone hydrology, and other uses that involve coupled thermal, hydrological, geochemical, and mechanical processes in permeable media.

The suite is continually being updated in response to scientific advancements, technical needs, user requests, and changes in hardware and software architectures. Foremost, code changes are made to provide the user with the capabilities needed to address specific scientific challenges. This requires the development of new equation-of-state (EOS) modules and refined process descriptions. Moreover, the simulators are being used for systems of increasing size and complexity, making high numerical performance a key target for further developments. The system of equations to be solved increases significantly, not only because of increased dimensionality, larger model domains, and higher resolution, which lead to many more gridblocks, but also because of the coupling of hydrological, thermal, mechanical, and biogeochemical processes, which leads to more equations that need to be solved per

gridblock. Finally, improved support tools for model setup, mesh generation, and visualization and analysis of modeling results are essential for making an advanced simulator applicable to complex subsurface flow and transport problems. This paper focuses on developments that are mainly concerned with numerical performance and code maintenance.

PERFORMANCE AND MAINTENANCE

Larger parallel computing resources are becoming economically viable for many TOUGH users. This increased processing power will naturally allow them to tackle problems with greater complexity and at a larger scale. In addition, the need to solve large problems is becoming more acute as numerical models are increasingly used as predictive and regulatory tools. For example, for a geological carbon sequestration problem that is of interest to government and industry alike, scientists and engineers have to use a numerical model that is sufficiently resolved over a large area, thus requiring the model to be solved in parallel. This parallelism is a driving requirement in the work described in this paper.

TOUGH was initially designed to be a serial code. To address the need to solve larger problems and to take advantage of the increased availability of multiprocessor computers, TOUGH2 (Pruess et al., 1999; Zhang et al., 2008) and TOUGH+ (Moridis et al., 2008) were parallelized based on the standard Message Passing Interface (MPI) protocol. iTOUGH2, which uses TOUGH runs to perform parameter estimation, is parallelized based on the Parallel Virtual Machine (PVM) protocol (Finsterle, 1998) and can run several (serial) forward simulations simultaneously.

While these efforts have made available parallel versions of TOUGH, this parallelization is far from comprehensive. First, the parallel versions of these codes are separate software packages with different source code and are not maintained or versioned with the "mainline" TOUGH codes. Second, other codes within the suite have not been parallelized at all. In particular, the reaction-transport code TOUGHREACT V2 is one of the most demanding codes in terms of equations solved and computing time, but it has only recently been partially parallelized for shared memory architectures using OpenMP (Sonnenthal et al., in prep.)

In addition, further work must be done on the existing parallelized codes to allow them to realize their full potential. Presently, these codes cannot perform parallel I/O tasks effectively. This can cause major slowdowns in simulations that write a lot of data. The parallel TOUGH codes are also unable to take advantage of new computing architectures such as heterogeneous multicore systems and GPU systems without significant code modifications.

There are several versions of TOUGH that were developed to solve particular problems (e.g., the TOUGH2 and TOUGH2-MP families, the TOUGH+ and pTOUGH+ families, as well individual codes such as TOUGHREACT, iTOUGH2, TOUGH-FLAC, etc.). While the development of a single inclusive "über-TOUGH" code may not be feasible (or desired), there are significant components of all TOUGH codes that are similar and could leverage a single shared library. There are obvious engineering benefits to having a common code base for the TOUGH applications; debugging and making incremental improvements becomes more practical when it can be done once, at a single location in the source code.

THE *TOUGHLIB* LIBRARY

A library called *toughlib* is being designed to provide an API that has functionalities common to all TOUGH codes. It will focus on two main functionalities: (1) to provide a common interface to all internal and external linear solvers, and (2) to provide a more user-friendly implementation of routines involving parallel communications. We believe the first functionality will have immediate benefits, since all TOUGH codes require solving a large linear system in the nonlinear solver used in codes. The second functionality will significantly reduce the overhead of maintaining (and debugging) a parallel code. It will also lower the barrier for future developers that are not familiar with parallel programming to more easily work with a parallel code.

toughlib consists of several Fortran90 modules. The modules contain data types that are designed in an object-oriented manner. The representation of a type, such as a sparse matrix or a vector, is thus separated from its implementation. Encapsulation of data and logic in a type reduces complexity of the code. *toughlib* also helps distinguish which data arrays are global or local to each processor.

Interface to External Linear Solver Packages

One important component to all TOUGH codes is the sparse linear solver. At present, the parallel versions of TOUGH (TOUGH-MP and pTough+) use AZTEC (Tuminaro et al., 1999) as the main workhorse. However, the development of AZTEC has long ceased, at least in its present form. As such, it does not include some of the latest developments of linear solver technologies. The serial version of the TOUGH codes (TOUGH2, TOUGHREACT and TOUGH+) use a collection of preconditioned iterative solvers implemented in T2SOLV (Moridis and Pruess, 1997), but this code has not been updated for the past 15 years.

To exploit some of the newer linear solver packages, *toughlib* will provide an interface to these packages. One solver that has demonstrated success on large parallel systems is PETSc (Portable, Extensible Toolkit for Scientific Computation; Balay et al., 2012). PETSc allows for the use of several preconditioners (e.g., LU, Block Jacobi, Multigrid) and has many other features.

While developers can work directly with the above packages, *toughlib* aims to provide a solver API that is specific to the TOUGH codes. It enables developers to only learn one interface and have access to all the linear packages that *toughlib* will incorporate. In addition, *toughlib* will also provide an API to the existing AZTEC and T2SOLV routines, allowing developers to easily transition to the new interface of *toughlib*. All the benefits of these additional external linear solvers will then be immediately available to developers, including the ability to choose the appropriate solvers and preconditioners, without recompiling the code.

In addition to the nonlinear solver used to solve the flow and transport phenomena, the geochemistry model in TOUGHREACT (Xu et al., 2011) requires solution of ordinary differential equations. While current in-house approaches appear to be satisfactory for the moment, *toughlib* can provide other alternatives. Under consideration is an interface to an external ODE solver such as the SUNDIALS package (Hindmarsh et al., 2005). This would allow TOUGHREACT users to easily test out different numerical algorithms to improve the efficiency of the code.

Figure 1 schematically shows the interfaces between *toughlib* and the core simulator.

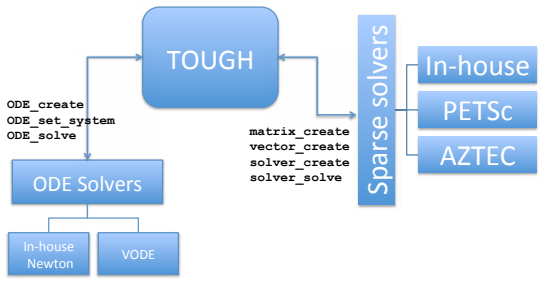

Figure 1. Interfaces are layers that allow interchangeable implementations.

Parallel Framework

High-performance computing is a peculiar branch of science, because its significance is completely determined by its success at performing ever-larger simulations in ever-less time. Thus, the performance of parallel simulations is paramount in any modern scientific code that aims to solve the types of problems that interest TOUGH users.

Improving the parallel performance first requires improving a parallel framework, accomplished by *toughlib*. The parallel versions of the TOUGH codes require several steps that are not needed in a serial version, such as domain decomposition, various functions required by the parallel solver algorithms, and any necessary additional data storage. Additionally, the serial versions can (and typically do) use a different solver than the parallel version. However, there are good reasons to write functions that will allow development of a TOUGH code that will run in serial or parallel (or parallel using only one processor). The functions in *toughlib* are

written such that they can be used in serial or parallel mode. For example, a function for domain decomposition would simply return when only one processor is being used. The advantages of one code that can run in serial or parallel are reduction in the total amount of code, easier maintenance, more robust testing, and easier debugging.

We would also like to make parallel programming with MPI easier for the TOUGH developer. One way to do this is write functions that do most or all of the communication in *toughlib* and provide convenience functions for doing typical operations, such as summing a variable to obtain the global value or interacting with the parallel file system.

There are various ways that *toughlib* can address parallel performance. If most of the code to handle parallel operations and communication are implemented in *toughlib*, then any performance improvements would benefit all TOUGH codes that use it. For example, a common performance improvement is caching and serializing data before sending it to another processor. It is often best to reduce the number of communication calls and increase the buffer sizes.

The field of high performance computation is advancing rapidly, and with each new parallel machine being built comes new hardware and software that may require code changes in the application to compile or effectively use the resources. The *toughlib* library will be portable and strive to be efficient across a wide range problem types and sizes on available platforms. It will focus on relatively uncomplicated data types such as matrices and vectors that are commonly used in existing TOUGH codes. and whose use and characteristics are well understood by developers.

Essential to improving computational performance for any serial or parallel code is the measurement of quantities like wall-clock time, memory use, and floating-point operations. Utility functions can help provide these in a manner that makes sense to TOUGH codes for various machine architectures.

Other *toughlib* Benefits

Toughlib uses CMake, which is a package for managing configuration and building of software applications. CMake is especially useful for building external libraries that *toughlib* or TOUGH codes may depend on. We will use version control software for developing *toughlib*, and it currently exists in a Mercurial repository (distributed source control management tool). The *toughlib* software will be distributed through a Bitbucket account at Lawrence Berkeley National Laboratory. Bitbucket offers secure source-code hosting of repositories, making it easier to manage team software development. CMake, Mercurial, and Bitbucket are freely available. These practices could be extended to the TOUGH codes.

We can include functions that perform routine checks, ensuring valid input to save time tracking down problems. Output functions, such as those for writing VTK-formatted files, could be maintained in *toughlib* as well. The mesh format for TOUGH codes has many features that are similar, and common operations on the mesh can be maintained in *toughlib*. Various physical constants could be defined and maintained in one place.

FUTURE WORK

TOUGHREACT developers are beginning to use threading (via OpenMP) to speed up the calculation of chemical reactions, which are usually the most cpu-intensive component of reactive-transport problems (Sonnenthal et al., in prep). This might prove to be beneficial for other TOUGH codes.

As *toughlib* matures, the TOUGH developers and users will be able to focus more on the science and less on the solvers and parallel concepts.

ACKNOWLEDGMENT

We acknowledge the support of Lawrence Berkeley National Laboratory's Innovation Grant. This work was supported, in part, by the U.S. Dept. of Energy under Contract No. DE-AC02-05CH11231.

REFERENCES

Finsterle, S., *Parallelization of iTOUGH2 Using PVM*, Report LBNL-42261, Lawrence Berkeley National Laboratory, Berkeley, Calif., October 1998.

Moridis, G.J., and K. Pruess, *T2SOLV: An Enhanced Package of Solvers for the TOUGH2 Family of Reservoir Simulation Codes*, Report LBNL-40933, Lawrence Berkeley National Laboratory, Berkeley, Calif., 1997.

Moridis, G.J., M.B. Kowalsky, and K. Pruess, *TOUGH+HYDRATE v1.0 User's Manual: A Code for the Simulation of System Behavior in Hydrate-Bearing Geologic Media*, Report LBNL-149E, Lawrence Berkeley National Laboratory, Berkeley, Calif., 2008.

Pruess, K., C. Oldenburg, and G. Moridis, *TOUGH2 User's Guide, Version 2.0*, Report LBNL-43134, Lawrence Berkeley National Laboratory, Berkeley, Calif., 1999.

Tuminaro, R. S., Heroux, M., Hutchinson, S. A., Shadid, J. N., *Official AZTEC User's Guide: Version 2.1*, December, 1999.

Balay S., Brown J., Buschelman K., Eijkhout V., Gropp W., Kaushik D., Knepley M., McInnes L., Smith B., and Zhang H., *PETSc Users Manual*, ANL-95/11 – Revision 3.3, Argonne National Laboratory, 2012.

Hindmarsh A. C., Brown P. N., Grant K. E., Lee S. L., Serban R., Shumaker D. E., and Woodward C. S., "*SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers*," ACM Transactions on Mathematical Software, 31(3), pp. 363-396, 2005.

Xu, T., N. Spycher, E. Sonnenthal, G. Zhang, L. Zheng, & K. Pruess, *TOUGHREACT Version 2.0: A simulator for subsurface reactive transport under non-isothermal multiphase flow conditions*.Computers and Geosciences, 37:763–774, 2011.

Zhang, K., Y.-S. Wu, and K. Pruess, *User's Guide for TOUGH2-MP — A Massively Parallel Version of the TOUGH2 Code*, Report LBNL-315E, Lawrence Berkeley National Laboratory, Berkeley, Calif., 2008.